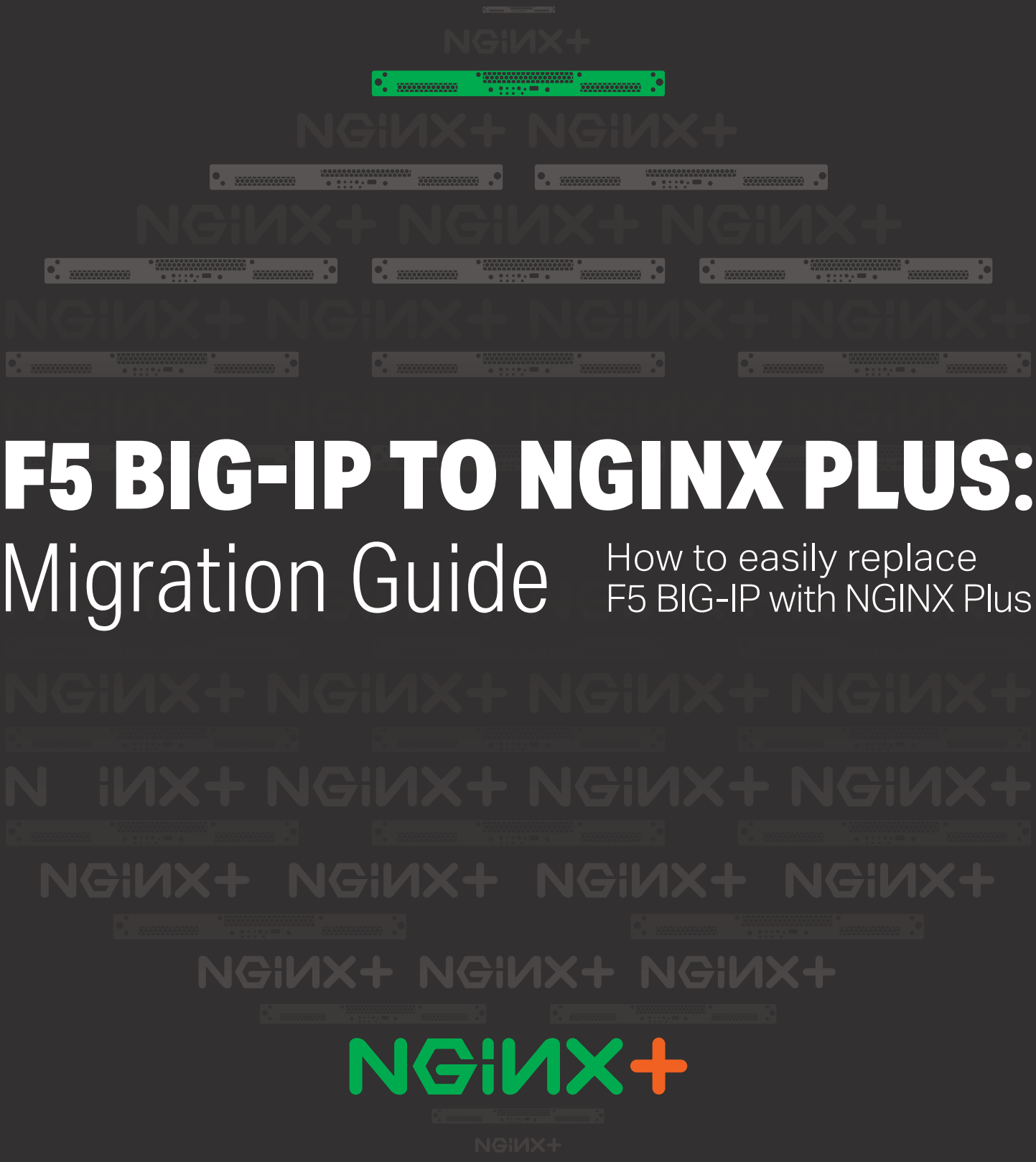


F5 BIG-IP TO NGINX PLUS: Migration Guide

How to easily replace F5 BIG-IP with NGINX Plus

NGINX+

F5 BIG-IP TO NGINX PLUS: Migration Guide

How to easily replace F5 BIG-IP with NGINX Plus

by Faisal Memon and Alan Murphy

NGINX

Why Replace FP BIG-IP with NGINX Plus?

Cost savings – Over 80% cost savings with NGINX Plus and industry standard x86 servers for the same or better performance.

Flexibility – Use NGINX Plus in any environment: public cloud, bare metal, containers, and virtual machines. No bandwidth or throughput limits.

Agility – Automate provisioning of new services. Deploy changes in seconds rather than weeks.

"It takes 2 weeks to get an F5 modification from the networking team. You know how long it takes us to change NGINX? It takes 30 seconds to make the change in GitHub, and then we run the Ansible script. Ta-da, production."

–Engineer at large telco company

Learn more at nginx.com

Table of Contents

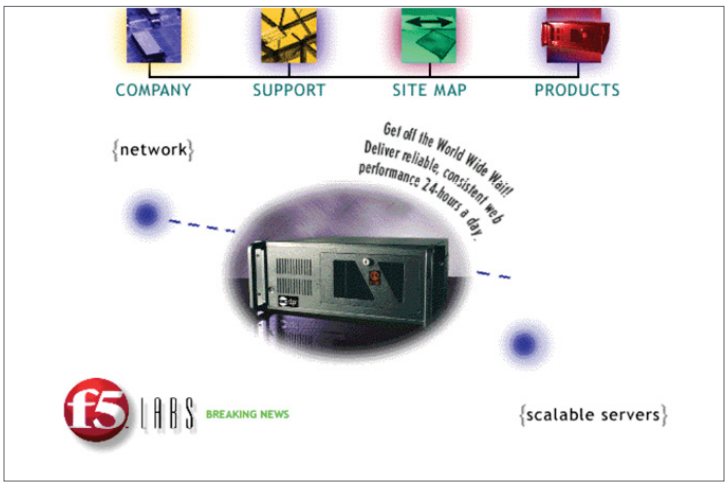
1 Introduction

"In a future that includes competition from open source, we can expect that the eventual destiny of any software technology will be to either die or become part of the open infrastructure itself."

–Eric S. Raymond, *The Cathedral and the Bazaar*, 1997

In the 1990s most websites were operating on just a single server. This led to a lot of instability and downtime, because the single server was a single point of failure (SPOF).

As websites and the Internet in general gained in popularity throughout the 90s, the only way for website owners to handle the additional load was to buy a bigger server, an extremely cost-prohibitive proposition, especially if they had already invested in an expensive server. At that time servers cost well over $50,000 and prices increased sharply for better performance.



F5.com, circa 1997. Source: www.archive.org

In 1996, Jeff Hussey founded F5 Labs (now F5 Networks) to solve both the SPOF and expense problems. In 1997 the company released the BIG/ip (now BIG-IP), a load balancer capable of distributing user traffic among multiple servers. This gave website owners another option for scaling their websites. Rather than buying a single bigger server they could build out an array of cheaper servers and frontend them with the cheaper BIG/ip which cost around $20,000. The result was improved performance and reliability at a reasonable cost.

The Rise of F5

With a strong value proposition in place, F5 took off. In June 1999 they filed their IPO (ticker symbol: FFIV) for $10/share. By November 1999 they were trading at $72.25/share. However, F5's customer base was almost entirely web startups, so it was hit hard by the dot-com bust: by April 2001 the stock price fell all the way to $2.28.

Hussey brought in industry veteran John McAdam to turn the company around. McAdam changed the focus of F5 away from dot coms toward larger brick-and-mortar enterprises. Under McAdam, F5 went from being 80% reliant on dot-com customers to 90% reliant on large enterprises.

The strategy paid off and by 2003 F5 was profitable and experiencing double digit growth. By January 10, 2011 FFIV was trading at $144.17. In both 2010 and 2011, F5 was on Fortune's list of 100 Fastest-Growing Companies. Also in 2010, F5 was one of the ten best-performing stocks on the S&P 500. Throughout the 2000s F5 completed numerous acquisitions to complement and expand the F5 BIG-IP feature set.

Five Industry Trends Are Disrupting F5 Today

Today, the IT landscape has changed. Servers are cheaper and ever more powerful. Software load-balancing solutions can handle the great majority of use cases, and are much faster and easier to configure than custom hardware.

F5 is stuck in a classic innovator's dilemma; they need to charge higher prices per unit as their ales slump, and they need to price their software solutions at a premium so as to limit the impact on their hardware sales. As a result, F5 BIG-IP sales are declining. In Q3 2018, F5 reported that systems revenue declined 1.5% year over year.[1] At one point, F5 hired Goldman Sachs to field takeover offers.[2] It seems clear that their current business model is unsustainable.

This will only get worse as five industry trends are accelerating pressure on F5.

F5 BIG-IP is unnecessarily expensive, with costs driven up by:

- **Complex licensing** – TF5 BIG-IP limits bandwidth, SSL transactions, and features in both hardware and virtual appliances. Customers are forced to pay for license upgrades to unlock the additional performance and features.
- **Custom hardware** – The F5 BIG-IP is a custom-built hardware appliance with ASICs, FPGAs, and hardware SSL accelerators, depending on the model. Custom hardware is expensive to create and most deployments do not need the extra functionality provided by the custom hardware deployments.
- **Legacy features** – A January 2018 O'Reilly survey found that 90% of hardware load balancer customers only use the basic functionality. This is especially true of F5 BIG-IP, which has accumulated many features over the years. Most are not used by average users, or even most power users, though every customer still pays for them.

1. https://www.zacks.com/stock/news/313725/f5-networks-ffiv-q3-earnings-and-revenues-top-estimates
2. https://www.reuters.com/article/us-f5networks-goldman-m-a-exclusive-idUSKCN0YT2JT

Trend 1: DevOps

F5 BIG-IP is used by many organizations as the ingress and egress point for all their applications. These can include the corporate website, internal-facing applications, external-facing applications, the corporate email server, etc.

As a result, BIG-IP is typically locked down and managed by the IT team, because a misconfiguration could make email and other critical business applications inaccessible to the entire company. In most companies, change requests for the F5 BIG-IP server are put in a queue and implemented slowly and carefully.

This does not mesh well with application teams and agile DevOps practices. For an application team to push out a new application version usually requires a change to the F5, to point to a new IP address and port. To request the change, the application team typically files an IT ticket, which is manually intensive and can take weeks to complete.

To get around this delay, many application teams are deploying NGINX directly in front of the application, its IP address and port number as the application's stable entry point, and then having the F5 BIG-IP point to that IP/port. The application team can now modify the application without involving IT. NGINX becomes a DevOps-friendly abstraction layer between the legacy F5 BIG-IP and the dynamic, fast-changing needs of application teams.

Trend 2: Public Cloud

In 2006, Amazon launched Amazon Web Services. Though AWS was not the first public cloud environment, Amazon's reach and marketing ability made it the first one to gain mass adoption. The public cloud has made it easier for companies to get started building web applications. You no longer have to purchase servers or space in a colocation facility; you just pay by the hour for compute and storage resources.

Public cloud stats

- Gartner expects public Infrastructure as a Server (IaaS) revenue to grow to $70 billion by 2020.
- AWS revenue grew 49%, to $5.44 billion, in Q1 2018 compared to Q1 2017. Google Cloud Platform and Microsoft are also gaining mainstream traction.
- O'Reilly surveyed 2,500 IT professionals and found that 54% have lowered IT operating costs by moving to a public cloud.

For a hardware vendor like F5, the public cloud is a problem. Though F5 does have an offering in AWS, it is priced higher than their hardware appliances so as not to cannibalize hardware sales. Customers of F5 in AWS are, in effect, paying for hardware they can't possibly use at all. In fact, the F5 cloud offering is over 7 GB in size, it is quite literally a copy of their hardware appliance and not cloud-native sotware. Making things worse, F5 limits their AWS offering to 5 Gbps in bandwidth, and have graded bandwidth limits starting at 200 Mbps, forcing you to pay more as your traffic increases.

In contrast, NGINX Plus is a cloud-native software solution available in all public cloud environments, including Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure. In true cloud-native fashion NGINX Plus is available with no bandwidth limits, so you are not forced to pay more if your traffic increases.

Trend 3: Commodity Servers

The price of standard x86 servers has steadily decreased, while performance has steadily increased. The price of F5 hardware has remained relatively flat. Whereas F5 was a cost-saving proposition when initially released, as it was priced to be cheaper than an x86 server, it is now cost prohibitive: 6-7x the cost of a commodity server that offers the same level of performance.

Rather than pay for custom hardware appliances, companies can instead use a cheap array of commodity servers running software solutions. Such solutions are not only cheaper, but also make it faster and easier to recover from a hardware failure. If an F5 BIG-IP fails, only F5 can provide a replacement. If a commodity server fails, another server in the pool can be used or procured from any supplier if need be.

NGINX Plus runs on all commodity servers. x86, ARM, and PowerPC are all supported.

NGINX facts:

- Used by more than 300 million websites as of August 2018
- Used by 64% of the top 10,000 busiest websites
- Used by 57% of the top 100,000 busiest websites
- Over 1 billion pulls on Docker Hub
- 1,500 NGINX Plus customers
- $43 million Series C round funding from Goldman Sachs and NEA

Trend 4: Open Source Software

In the next 20 years, it will be nearly impossible to develop large-scale software or platforms without using open source software. Most technology innovations we witness today are built on the back of open software. Even the F5 BIG-IP's operating system is built on the open source Linux operating system.

Open source alternatives are available for many commercial products, giving organizations low-to-no cost alternatives to proprietary solutions. Open source alternatives to the BIG-IP started appearing in the 2000s, and continue to be created and improved upon today. NGINX is one of the many open source competitors to F5 in the proxy and load-balancing space. Running on commodity hardware, both provide most of the F5 feature set at a far lower cost.

Trend 5: Microservices

Microservices, or distributed applications, are the future of application development. They're a departure from the standard monolithic way of developing applications where all functionality is combined into a single large binary. Instead, in microservices-based applications functionality is split up into self-contained services which each expose an API. Communication between services happens through the API. Structuring applications this way increases agility by enabling development teams to work independently on their microservices, rather than having to coordinate on a shared monolith.

Microservices applications are more dynamic than monolithic applications. In addition they generate east-west traffic (services communicating with one another) which monolithic applications don't need to. These requirements put new demands on the load-balancing infrastructure. For example, when a new version of a microservice is deployed, the load balancer needs to automatically detect this and route traffic to the new microservice. With F5, a human administrator has to make the necessary configuration change manually with the UI. With NGINX Plus this process can easily be fully automated. Additionally NGINX Plus itself can run inside of a container; F5 has no container option

Why Software Load Balancing

Over the long run, choosing a software-based load balancer saves tremendous time, money, and effort, while resulting in applications designed for optimal performance. The business as a whole benefits from this approach; the better applications perform, the more successful the business becomes.

Here are five reasons to move from traditional hardware ADCs like F5 BIG-IP to software solutions:

1. **Dramatically reduce costs** without sacrificing features or performance. You can save roughly 85% for the same price/performance with NGINX Plus vs. F5 BIG-IP.
2. **Gain agility** by enabling modern development and delivery frameworks, like DevOps. As applications move to continuous integration/continuous delivery (CI/CD), and as teams restructure to make rapid deployments possible, waiting days or weeks for an ops team to reconfigure a hardware ADC is unimaginable. Only software provides the rapid configuration, flexibility, and application-level control that DevOps requires.
3. **Simplify your architecture** by deploying one ADC solution everywhere. As software, NGINX Plus works the same way on all platforms – in VMs or containers, on premises, and on public, private, or hybrid cloud – making deployment flexible and easy.
4. **Adapt quickly** to changing demands on your applications. No waiting for special hardware, installation, and configuration when traffic is rising. It takes just minutes to install and configure NGINX to immediately scale up or scale out your applications to respond in real time to the demands on them.
5. **Improve customer experience** by eliminating artificial or contract-driven constraints on performance. Hardware ADC vendors usually build throughput caps into your contract, then hit you with hefty upgrade fees when traffic breaks through that cap. Software-based solutions have no artificial limits – simply upgrade the underlying hardware or virtualized compute power to unlock greater performance. Say goodbye to hefty fees for traffic increases.

Why Open Source?

Whether or not we realize it, we interact with open source software on a daily basis. Apple MacOS is based on the open source FreeBSD. Amazon Elastic Compute Cloud (EC2) is built on the open source Xen hypervisor. Even newer Cisco products (such as the Cisco Nexus line of switches) use Linux underneath the hood.

The technological innovation we benefit from today is only possible because of open source. People freely sharing ideas and code has enabled us to move much faster than if we all stayed in our silos not collaborating or sharing. In the next 20 years, it will be nearly impossible to develop large-scale software or platforms without using open source software.

Open source software is free and easily attainable, providing great benefits to the business. Rather than waiting weeks to procure a custom hardware appliance, anyone can download open source software in minutes.

The best part of open source is the community. NGINX, for example, is used by more than 250 million websites according to the July 2018 Netcraft web server survey. As a result there is a community around NGINX, providing help, guidance, and tip and tricks to other users. The only way to get that reach is with open source.

Brief History of NGINX

Igor Sysoev, NGINX creator

The open source NGINX software was created by Igor Sysoev as a side project while he was working as a sysadmin at Rambler, a Russian service provider similar to Yahoo!. While at Rambler, Igor was asked to look into enabling the Apache HTTP servers to better handle the influx of traffic the company was receiving.

Apache HTTP Server is a web server, a website component that is responsible for delivering static assets, such as images, to users. While looking for ways to improve Apache's performance, Igor found himself blocked by several inherent design choices that hampered Apache's ability to handle 10,000 simultaneous users, commonly known as the C10K problem. In the spring of 2002 Igor started developing NGINX with an event-driven architecture that addressed the shortcomings in Apache.

On October 4th, 2004, Igor publicly released the source code of NGINX for free. From there NGINX quickly gained in popularity among web developers who immediately benefit from its improved performance over Apache.

> *"I wanted people to use it, so I made it open source."*
>
> –Igor Sysoev, NGINX creator

Seven years later, Igor and co-founders Andrew Alexeev and Maxim Konovalov created NGINX, Inc. and in 2013 the first version of NGINX Plus, the commercial offering, was released. NGINX Plus builds on the popular web server by adding load balancing and other advanced ADC features.

Today NGINX, Inc. is a global company with offices in San Francisco, Moscow, Singapore, and Cork. NGINX, Inc. has over 200 employees and 1,200 customers. It is venture-capitalist (VC) funded with over $100 million in funding to date from firms such as eVentures, Goldman Sachs, NEA, and Telstra.

2 Prepare Your Server

"It is said that the only things certain in life are death and taxes. For those of us in the IT industry, we can add one more to the list: commoditization."

–Ian Murdock, Creator of Debian Linux

One of the biggest benefits of F5 BIG-IP hardware appliances is that all the work of setting up a server is done for you, including all software installation. When using open source software, or software solutions in general, you have to prepare the server yourself. For NGINX or NGINX Plus, this includes purchasing a server of the right size, installing Linux, and finally installing NGINX or NGINX Plus.

In this chapter we walk you step-by-step through the process of installing NGINX or NGINX Plus, to make it as easy as possible. By the end of the chapter you will know how to prepare a server. You may even prefer it over the turnkey hardware appliance, because there is much more room for customization.

Perfect forward secrecy

Modern SSL standards mandate perfect forward secrecy (PFS), which ensures that encrypted traffic captured at a certain time (in a man-in-the-middle attack, say) can't be decrypted later (at the hijacker's leisure), even if the private key is compromised. PFS is recommended for maximum protection of user privacy in the current security climate.

PFS is more computationally expensive and so reduces TPS. As a result most vendors do not publish TPS data for PFS; keep this in mind when comparing specifications from vendors.

Purchasing a Commodity Server

There are many benefits to commodity servers. For example, if something fails, it is much easier and quicker to bring up a new x86 server than to procure a new hardware appliance. It's also easier to right-size the solution for your application needs, rather than buying a hardware appliance that is less customizable. These benefits, coupled with the low cost of commodity servers, drives the cost of the ADC to a fraction of an F5 BIG-IP.

NGINX runs on commodity x86 servers that you can purchase from Dell, HPE, Lenovo, SuperMicro and other vendors. These servers do not have built-in hardware SSL acceleration, but thanks to the drastic increase in compute power due to Moore's law, x86 servers using software SSL can provide enough performance for most deployments.

FP BIG IP model	Equivalent x86 Server Specs	X86 Server Cost
i2600	• 8 CPU cores • 4 GB RAM • 2x10 Gbe NIC • 1 TB HDD	$2,200 (88% savings)
i2800	• 16 CPU cores • 8 GB RAM • 2x10 Gbe NIC • 1 TB HDD	$3,500 (88% savings)
i4600	• 16 CPU cores • 8 GB RAM • 2x10 Gbe NIC • 1 TB HDD	$3,500 (88% savings)
i4800	• 32 CPU cores • 8 GB RAM • 2x10 Gbe NIC • 1 TB HDD	$6,000 (87% savings)
i5600	• 32 CPU cores • 8 GB RAM • 2x10 Gbe NIC • 1 TB HDD	$6,000 (89% savings)

A single NGINX Plus instance running on a high-end x86 server can achieve 1.2M HTTP requests per second, 61k SSL transaction per second and 70 Gbits of throughput. If you wish to specify a cluster that can handle more than this level of traffic, you can deploy NGINX Plus in a multiple-active, multiple-redundant fashion.

FP BIG IP model	Equivalent x86 Server Specs	X86 Server Cost
i5800	2 of the following: • 32 CPU cores • 8 GB RAM • 2x10 Gbe NIC • 1 TB HDD	$12,000 (82% savings)
i7600	2 of the following: • 32 CPU cores • 8 GB RAM • 2x40 Gbe NIC • 1 TB HDD	$12,000 (84% savings)
i7800	3 of the following: • 32 CPU cores • 8 GB RAM • 2x40 Gbe NIC • 1 TB HDD	$18,000 (81% savings)
i10600	2 of the following: • 32 CPU cores • 8 GB RAM • 2x40 Gbe NIC • 1 TB HDD	$12,000 (88% savings)
i10800	3 of the following: • 44 CPU cores • 16 GB RAM • 4x40 Gbe NIC • 1 TB HDD	$30,000 (74% savings)
i11600	2 of the following: • 44 CPU cores • 16 GB RAM • 4x40 Gbe NIC • 1 TB HDD	$30,000 (85% savings)

Note: NGINX is also supported on ARM and Power8 servers for some types of Linux.

Installing Linux

If your experience is with hardware appliances and you've have never used Linux before, installing it may seem daunting. It's not easy to learn a new operating system (OS). But if you've figured out Cisco IOS command syntax, you're more than capable of learning Linux! Plus, Linux is becoming the de facto industry-standard OS in modern IT environments, so learning it is mandatory for anyone continuing to work in the industry.

In this section we explain how to pick a Linux distribution and install it.

Choosing the Right Distro

There are many Linux distributions (distros) and versions of each, which may add to the confusion. Of the many Linux distros, we recommend either of the following:

- Ubuntu Server 16.04 LTS or later
- CentOS or Red Hat Enterprise Linux (RHEL) 7.4 or later (CentOS is the community edition of RHEL)

Note: The remainder of this migration guide will assume you are using one of these operating systems.

These distributions all come with version 1.0.2 of the OpenSSL library. OpenSSL is the software that handles SSL/TLS operations in Linux. Version 1.0.2 performs 2 to 3 times better than version 1.0.1 (found in older Linux distros). OpenSSL 1.0.2 is also a requirement for HTTP/2, the latest version of the HTTP standard which also improves performance.

NGINX supports Ubuntu Server 16.04 LTS on x86, ARM, and Power8 servers, and CentOS/RHEL 7.4+ on x86 and Power8 servers.

Professional support is available for both Ubuntu Server and RHEL, from Canonical and Red Hat respectively.

Installing Linux

Before installing Linux, you need two things:

- A USB memory stick with 4 GB or more
- A server with connectivity to a network with DHCP and Internet access, both required for the Linux installer to retrieve packages over the Internet.

After you meet the prerequisites, there are just five steps:

1. Download the .iso file for CentOS 7.4+, RHEL 7.4+, or Ubuntu Server 16.04 LTS.

   You can download CentOS and Ubuntu Server for free; for RHEL you must first purchase a subscription or start an evaluation.

2. Create a bootable USB stick, the easiest way to install Linux. Ubuntu provides instructions for Mac and Windows, but they apply to CentOS and RHEL as well – just substitute the appropriate **.iso** filename.

3. Boot up the server with the bootable USB stick. This is the default boot process on most servers but if that is not the case, check the boot order settings in your BIOS.

4. Follow the wizard to install Linux. This is normally straightforward, requiring you only to set the date, time, and time zone. Be sure not to install the GUI if that option is offered.

5. Configure a static IP address. There are separate instructions below for CentOS/RHEL and Ubuntu Server.

Configuring a Static IP Address on CentOS and RHEL

To configure a static IP address on CentOS and RHEL we will use the Network Manager.

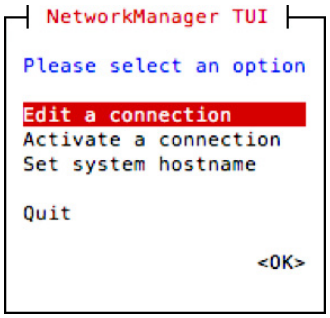
1. Use **nmcli** to list the available network interfaces:

```
$ nmcli
ens33: connected to ens33
    "Intel 82545EM Gigabit Ethernet Controller (Copper) (PRO/1000 MT
    Single Port Adapter)"
    ethernet (e1000), 00:0C:29:18:91:6A, hw, mtu 1500
    ip4 default
    inet4 192.168.179.232/24
    inet6 fe80::4978:2423:e3e6:536f/64
```

The **inet4** field lists the current IP Address of the interface. The name of the interface in this example is **ens33**. The value varies by system and is based on multiple factors such as the PCI Express hotplug slot index number. If there are multiple interfaces they will all be listed here.

2. Bring up the Network Manager test-based user interface (TUI):

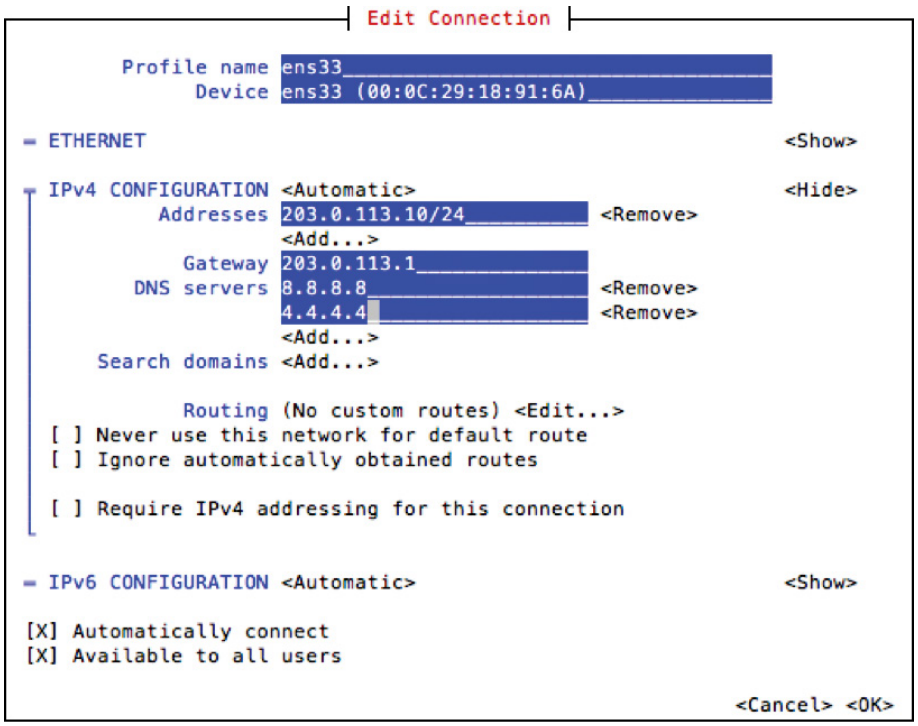
```
$ sudo nmtui
```



3. Select **Edit** a connection and then select the appropriate interface in the list.

4. In the **Edit** Connection menu, change IPv4 Configuration from **Automatic** to **Manual** and the press Enter on **<Show>** to show the current IP settings.

5. Modify the IP settings for your environment



6. Press **<OK>** to save the settings, and the press Esc twice to exit out of nmtui.

7. Restart the networking service for the changes to take effect.

```
$ sudo systemctl restart network
```

Configuring a Static IP Address on Ubuntu Server Linux

To configure a static IP address on Ubuntu Server Linux, you use a text editor to edit the **/etc/network/interfaces** file, changing the default DHCP setting to add some information about the static IP address that you want to configure. Here we're using the nano text editor.

1. Open **/etc/network/interfaces** in the text editor.

```
$ sudo nano /etc/network/interfaces
```

2. Search for "`# The primary network interface`". The two lines after it look something like this:

```
# The primary network interface
auto ens33
iface ens33 inet dhcp
```

   The name of the interface in this example is **ens33**. The value varies by system and is based on multiple factors such as the PCI Express hotplug slot index number.

3. Comment out the line that ends with dhcp by putting a **#** in front of it, then add the following lines so that the complete section looks like this:

```
# The primary network interface auto ens33
# iface ens33 inet dhcp
iface ens33 inet static
    address 203.0.113.10
    netmask 255.255.255.0
    gateway 203.0.113.1/24

dns-nameservers 8.8.8.8 8.8.4.4
```

Replace the two instances of the server's IP address (203.0.113.1) and the IP addresses for the DNS name server (8.8.8.8 and 8.8.4.4) with the appropriate values for your server. This example refers to Google's public DNS servers.

4. Save and close the file. With nano, press Ctrl+x to exit and enter y to save. Press Enter again to overwrite the current file.
5. Normally you do not need to reboot to update network settings. You can just run this command to restart the networking service:

```
$ sudo /etc/init.d/networking restart
```

If that does not work, restart Linux:

```
$ sudo restart now
```

Useful Linux commands:

`ls -lF` – List all files including file sizes

`rm` *`file`* – Remove a file

`cp` *`file1 file2`* – Copy file1 to file2

`cat` *`file`* – View the contents of a file

`mkdir` *`directory`* – Create a new directory

`rm -rf` *`directory`* – Forcefully remove a directory and all contents

`pwd` – Display the present working directory

`top` – Display top processes

`ps -ef` – Display all currently running processes on the system, including process id (PID)

`ps -ef | grep` *`processname`* – Display process info for processname

`kill pid` – kill process with process ID of *pid*

Installing NGINX or NGINX Plus

Now that you've installed Linux, the next step is to install NGINX or NGINX Plus. The installation steps differ for NGINX and NGINX Plus, and further vary depending on the operating system you are using. We're providing instructions for both NGINX and NGINX Plus on both reference OSs.

NGINX Plus is available at per-instance subscription prices for small deployments and at special package rates for larger deployments. For details, see our pricing page. You can also try NGINX Plus for free for 30 days or buy NGINX Plus online.

Once your subscription or free trial begins, NGINX, Inc. will send and instructions for logging in at the NGINX Plus Customer Portal.

Installing Open Source NGINX on CentOS and RHEL

1. Add the official NGINX repository to your `yum` repositories. Run the following command:

```
$ sudo echo \
"[nginx]
name=nginx repo
baseurl=http://nginx.org/packages/OS/7/\$basearch/
gpgcheck=0
enabled=1" > /etc/yum.repos.d/nginx.repo
```

   Replace *`OS`* with `centos` or `rhel` as appropriate. If you are using CentOS/RHEL 6.5+, change 7 to 6.

2. Install NGINX:

```
$ sudo yum update && sudo yum install nginx
```

3. Start NGINX and set it to start automatically on reboot:

```
$ sudo systemctl enable nginx
$ sudo systemctl start nginx
```

4. Enable incoming traffic to port 80, because by default CentOS and RHEL block all incoming connections. If NGINX will be listening on other ports (for example, on port 443 for HTTPS), repeat the first **`firewall`** command for each one:

```
$ sudo firewall-cmd --permanent --zone=public --add-port=80/tcp
$ sudo firewall-cmd --reload
```

Installing Open Source NGINX on Ubuntu Server Linux

1. Add the NGINX signing key to the **`apt`** program keyring:

```
$ wget -qO - https://nginx.org/keys/nginx_signing.key | sudo
apt-key add -
```

2. Add the official NGINX repository to your apt repositories. Run the following command:

```
$ sudo echo \
"deb http://nginx.org/packages/mainline/ubuntu/ codename nginx
deb-src http://nginx.org/packages/mainline/ubuntu/ codename nginx" \
| sudo tee /etc/apt/sources.list.d/nginx.list
```

Replace ***`codename`*** with **`xenial`** for Ubuntu 16.04 or **`zesty`** for Ubuntu 17.04.

3. Install NGINX:

```
$ sudo apt-get update && sudo apt-get install -y nginx
```

4. Start NGINX (it will automatically start on reboot):

```
$ sudo /etc/init.d/nginx start
```

Installing NGINX Plus on CentOS and RHEL

1. Create the /etc/ssl/nginx directory:

```
$ sudo mkdir /etc/ssl/nginx
$ cd /etc/ssl/nginx
```

2. Log in to NGINX Plus Customer Portal.

3. Download your version of *nginx-repo.crt* and *nginx-repo.key* files from NGINX Plus Customer Portal and copy the files to **/etc/ssl/nginx/** directory:

```
$ sudo cp nginx-repo.crt /etc/ssl/nginx/
$ sudo cp nginx-repo.key /etc/ssl/nginx/
```

4. Install **ca-certificates**:

```
$ sudo yum install ca-certificates
```

5. Download the **nginx-plus-repo** file and copy it to the **/etc/yum.repos.d/** directory:

```
$ sudo wget -P /etc/yum.repos.d \
https://cs.nginx.com/static/files/nginx-plus-7.4.repo
```

For CentOS/RHEL 7.0-7.3:

```
$ sudo wget -P /etc/yum.repos.d \
https://cs.nginx.com/static/files/nginx-plus-7.repo
```

For CentOS/RHEL 6:

```
$ sudo wget -P /etc/yum.repos.d \
https://cs.nginx.com/static/files/nginx-plus-6.repo
```

6. Install NGINX Plus:

```
$ sudo yum install nginx-plus
```

7. Start NGINX and set it to start automatically on reboot:

```
$ sudo systemctl enable nginx
$ sudo systemctl start nginx
```

8. Enable incoming traffic to port 80, because by default CentOS and RHEL block all incoming connections. If NGINX will be listening on other ports (for example, on port 443 for HTTPS), repeat the first **`firewall`** command for each one.

```
$ sudo firewall-cmd --permanent --zone=public --add-port=80/tcp
$ sudo firewall-cmd --reload
```

Installing NGINX Plus on Ubuntu Server Linux

1. Create the /etc/ssl/nginx directory:

```
$ sudo mkdir /etc/ssl/nginx
$ cd /etc/ssl/nginx
```

2. Log in to NGINX Plus Customer Portal.

3. Download your version of *nginx-repo.crt* and *nginx-repo.key* files from NGINX Plus Customer Portal and copy the files to **/etc/ssl/nginx/** directory:

```
$ sudo cp nginx-repo.crt /etc/ssl/nginx/
$ sudo cp nginx-repo.key /etc/ssl/nginx/
```

4. Download the NGINX signing key from nginx.org and add it:

```
$ wget -qO - https://nginx.org/keys/nginx_signing.key | \
sudo apt-key add -
```

5. Install **apt-utils** package and install NGINX Plus repository:

```
$ sudo apt-get install apt-transport-https lsb-release \
ca-certificates
$ printf "deb https://plus-pkgs.nginx.com/ubuntu 'lsb_release -cs' \
nginx-plus\n" | sudo tee /etc/apt/sources.list.d/nginx-plus.list
```

6. Download the ***90nginx*** file to **/etc/apt/apt.conf.d**:

```
$ sudo wget -qO /etc/apt/apt.conf.d/90nginx \
https://cs.nginx.com/static/files/90nginx
```

7. Install NGINX Plus:

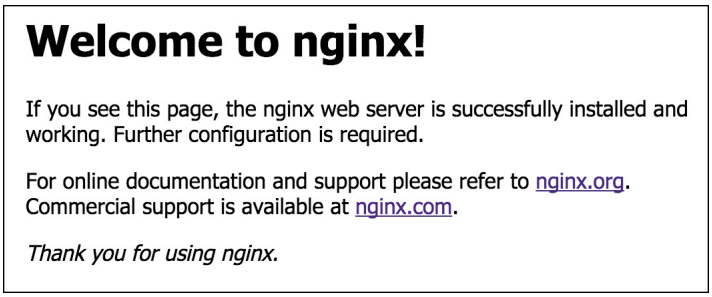
```
$ sudo apt-get update & sudo apt-get install -y nginx-plus
```

8. Start NGINX (it will automatically start on reboot):

```
$ sudo /etc/init.d/nginx start
```

Verifying Installation

For both NGINX and NGINX Plus, you can verify successful installation by navigating in a browser to the hostname or IP address of the NGINX or NGINX Plus server. The "Welcome to nginx!" page confirms that the server is running:



If you can access the "Welcome to nginx!" page, you've sucessfully installed NGINX

3 Migrate F5 BIG-IP Configuration to NGINX Plus

"The market is moving away from [F5], and is not coming back."

–Trip Chowdry, Analyst, Global Equities Research[1]

NGINX Plus provides a flexible replacement for traditional hardware-based application delivery controllers (ADCs). NGINX Plus is a small software package that can be installed just about anywhere – on bare metal, a virtual machine, or a container, and on-premises or in public, private, and hybrid clouds – while providing the same level of application delivery, high availability, and security offered by legacy ADCs. This guide explains how to migrate an F5 BIG-IP Local Traffic Manager (LTM) configuration to the NGINX Plus software application delivery platform, and covers the most commonly used features and configurations to get you started quickly on your migration.

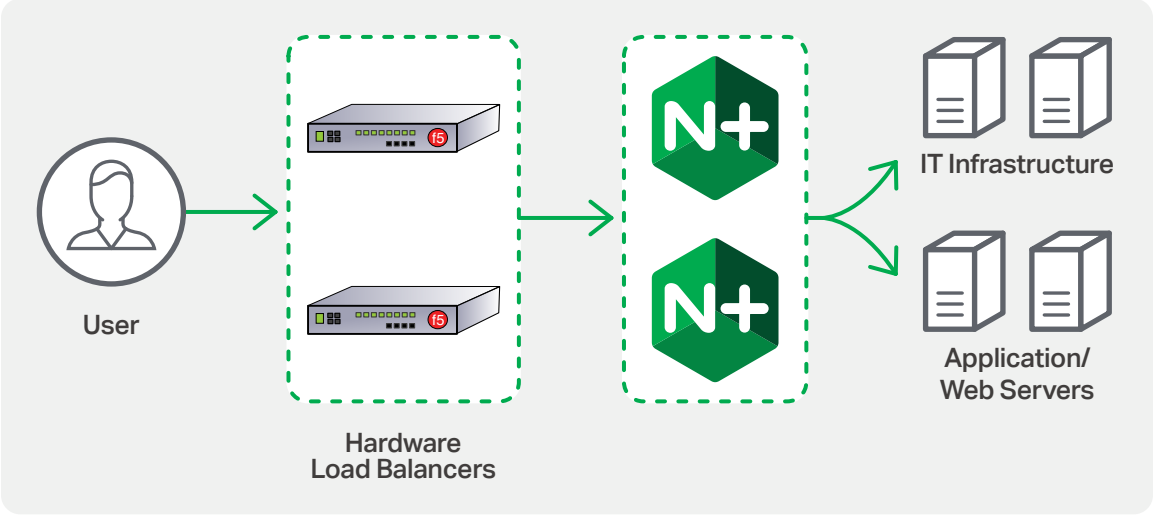
NGINX Plus and BIG-IP LTM both act as a full reverse proxy and load balancer, so that the client sees the load balancer as the application and the backend servers see the load balancer as the client. This allows for great control and fine-grained manipulation of the traffic. This guide focuses on basic load balancing. For information on extending the configuration with Layer 7 logic and scripting, see the post about migrating Layer 7 logic on the NGINX blog. It covers features such as content switching and request routing, rewriting, and redirection.

1. https://whotrades.com/blog/43553397328

NGINX Plus Deployment Scenarios

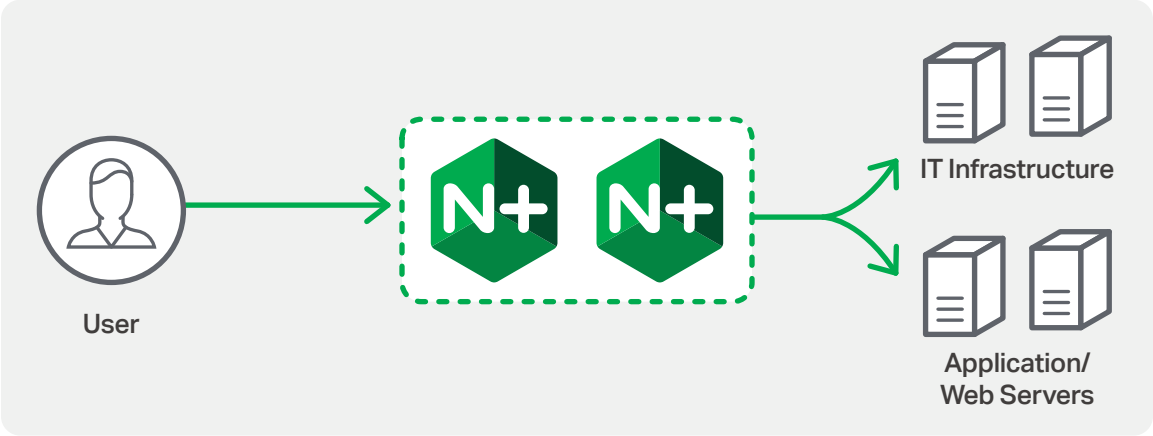
Architecturally speaking, NGINX Plus differs from traditional ADCs in deployment location and function. Typical hardware-based ADCs are usually deployed at the edge of the network and act as a front-door entry point for all application traffic. It's not uncommon to see a large hardware ADC straddle the public and private DMZs, assuming the large burden of processing 100% of the traffic as it comes into the network. You often see ADCs in this environment performing all functions related to traffic flow for all applications – security, availability, optimization, authentication, etc. – requiring extremely large and powerful hardware appliances. The downside to this model is that the ADC is always stationary at the "front door" of the network.

As they update their infrastructure and approach to application delivery, many companies are paring down the hardware ADC functionality at the edge and moving to a more distributed application model. Because the legacy hardware ADC is already sitting at the edge of the network it can continue to handle all ingress traffic management, directing application traffic to the appropriate NGINX Plus instances for each application type. NGINX Plus then handles traffic for each application type to provide application-centric load balancing and high availability throughout the network, both on- and off-premises. NGINX Plus is deployed closer to the application and is able to manage all traffic specific to each application type.



NGINX Plus can run behind BIG-IP LTMs to handle application traffic

Other companies are completely replacing their stationary hardware ADC appliances at the network edge with NGINX Plus, providing the same level of application delivery at the edge of the network.



NGINX Plus can completely replace hardware ADCs to handle all traffic entering the network

Prerequisites

This guide assumes you are familiar with F5 BIG-IP LTM concepts and CLI configuration commands. Familiarity with basic NGINX software concepts and directives is also helpful; links to documentation are provided, but the guide does not explain NGINX Plus functioning in depth.

Mapping F5 BIG-IP LTM Networking Concepts to NGINX Plus

Network Architecture

When migrating F5 BIG-IP LTM networking and load-balancer configuration to NGINX Plus, it can be tempting to try translating F5 concepts and commands directly into NGINX Plus syntax. But the result is often frustration, because in several areas the two products don't align very closely in how they conceive of and handle network and application traffic. It's important to understand the differences and keep them in mind as you do your migration.

F5 divides the network into two parts: the management network (often referred to as the management plane or control plane) and the application traffic network (the data plane). In a traditional architecture, the management network is isolated from the traffic network and accessible via a separate internal network, while the application network is attached to a public network (or another application network). This requires separate network configurations for each of the two kinds of traffic.

BIG-IP LTM appliances are a dual-proxy environment, which means that data plane traffic is also split between two different networks: the client-side network over which client requests come into the BIG-IP LTM, and the server-side network over which requests are sent to the application servers. BIG-IP LTM typically requires two network interface cards (NICs) to handle each part of the network.

It is possible with a BIG-IP LTM appliance, however, to combine the client and server networks on a single NIC, combining the data plane into a single-stack proxy architecture. This is a very typical architecture in a cloud environment where traffic comes into the BIG-IP LTM data plane and exits through the same virtual NIC. Regardless of networking architecture, the same basic principles for load balancing apply, and the configurations discussed below work in either architectural layout.

NGINX Plus can function in a similar architecture either by binding multiple IP subnets (and/or VLANs) to a single NIC that is available to the host device, or by installing multiple NICs and using each for unique client and server networks, or multiple client networks and multiple server-side networks. This is, in essence, how the BIG-IP LTM appliance functions as well, typically shipping with multiple NICs which can be trunked or bound into virtual NICs

Definitions of Networking Concepts

Basic F5 BIG-IP LTM networking configuration requires only that you specify the IP addresses of the management and data planes, but managing more complex network environments that include BIG-IP LTM appliances involves some additional concepts. All of these concepts can be very easily simplified and mapped to NGINX Plus instances. Key BIG-IP LTM networking concepts with NGINX Plus correlates include:

- **Self-IP address** – The primary interface that listens to incoming client-side data plane traffic on a specific VLAN. It is a specific IP address or subnet on a specific NIC associated with that VLAN or a VLAN group.

  In NGINX Plus, self IP addresses most directly map to the primary host interface used by NGINX Plus to manage traffic-plane application data. Generally speaking, self IP addresses are not a necessary concept in an NGINX Plus deployment, as NGINX Plus utilizes the underlying OS networking for management and data-traffic control.

- **Management IP address:port pairs** – The IP address:port combinations on a BIG-IP LTM appliance that are used to administer it, via the GUI and/or remote SSH access. The NGINX Plus equivalent is the Linux host IP address, typically the primary host interface. It is possible, but not necessary, to use separate IP addresses and/or NICs for management access to the Linux host where NGINX Plus is running, if you need to separate remote access from the application traffic.
- **Virtual server** – The IP address:port combination used by BIG-IP LTM as the public destination IP address for the load-balanced applications. This is the IP-address portion of the virtual server that is associated with the domain name of a frontend application (for instance), and the port that's associated with the service (such as port 80 for HTTP applications). This address handles client requests and shifts from the primary device to the secondary device in the case of a failover.

  Virtual servers in NGINX Plus are configured using a `server` block. The `listen` directive in the `server` block specifies the IP address and port for client traffic.
- **Pool** and **node list** – A *pool* is a collection of backend nodes, each hosting the same application or service, across which incoming connections are load balanced. Pools are assigned to virtual servers so BIG-IP LTM knows which backend applications to use when a new request comes into a virtual server. In addition, BIG-IP LTM uses the term *node list* to refer to an array of distinct services that all use the same traffic protocol and are hosted on the same IP address, but listen on different port numbers (for example, three HTTP services at 192.168.10.10:8100, 192.169.10.10:8200, and 192.168.10.10:8300).

  NGINX Plus flattens the BIG-IP LTM pool and node list concepts by representing that information in upstream configuration blocks, which also define the load-balancing and session-persistence method for the virtual server that forwards traffic to the group of backend servers. NGINX Plus does not need the concept of node lists, because standard `upstream` block configuration very easily accommodates multiple services on the same IP address.

In addition to these networking concepts, there are two other important technology categories to consider when migrating from BIG-IP LTM to NGINX Plus:

- **iRules** – iRules is a proprietary, event-driven, content-switching and traffic-manipulation engine (based on TCL) used by BIG-IP LTM to control all aspects of data-plane traffic. iRules are attached to virtual servers and are required for any type of content switching, such as choosing a pool based on URI, inserting headers, establishing affinity with JSESSIONIDs, and so on. iRules are event-driven and are configured to fire for each new connection when certain criteria are met, such as when a new HTTP request is made to a virtual server or when a server sends a response to a client.

  NGINX Plus natively handles content switching and HTTP session manipulation, eliminating the need to explicitly migrate most context-based iRules and those which deal with HTTP transactions such as header manipulation. Most context-based iRules can be translated to `server` and `location` blocks, and more complex iRules that cannot be duplicated with NGINX Plus directives and configuration block can be implemented with the Lua or JavaScript modules. For more information on converting iRules to NGINX Plus configuration, see Chapter 4.

- **High availability** – Conceptually, BIG-IP LTM and NGINX Plus handle high availability (HA) in the same way: each active-passive pair of load balancers shares a floating "virtual" IP address (VIP) which maps to the currently active instance. If the active instance fails, the passive instance takes over and assumes the VIP.

  BIG-IP LTM uses a built-in HA mechanism to handle the failover.

  For on-premises deployments, NGINX Plus uses a separate software package called **nginx-ha-keepalived** to handle the VIP and the failover process for an active-passive pair of NGINX Plus servers. The package implements the VRRP protocol to handle the VIP. Limited active-active scenarios are also possible with the **nginx-ha-keepalived** package. NGINX Plus also shares state in a cluster using the zone_sync configuration directive.

  Solutions for high availability of NGINX Plus in cloud environments are also available, including these:

  – Active-Active HA for NGINX Plus on AWS Using AWS Network Load Balancer

  – Active-Passive HA for NGINX Plus on AWS Using Elastic IP Addresses

  – All-Active HA for NGINX Plus on the Google Cloud Platform

Converting F5 BIG-IP LTM Load-Balancer Configuration to NGINX Plus

F5 BIG-IP LTM offers three methods for configuration:

- GUI
- CLI (the custom on-box Traffic Management Shell [TMSH] tool)
- iControl API

Ultimately all changes made via the GUI or API are translated to a TMSH CLI command, so that's the representation we're using in this guide. We assume that you are configuring the device from the (`tmos.ltm`) location, and so omit the common command variable `ltm` from all of the `TMSH` commands.

With NGINX Plus, configuration is stored in a straightforward text file which can be accessed directly or managed using traditional on-box tools or configuration management and orchestration tools such as Ansible, Chef, and Puppet.

Although the examples in this guide use only IP addresses to identify virtual servers, with NGINX Plus both the listening IP address:port combination and the `Host` header can be used to select the appropriate `server` block to process a request. Include the `server_name` directive in that block to specify the values to match in the Host header.

The list of parameters to the `server_name` directive can include multiple hostnames, wildcards, and regular expressions. You can include multiple `server_name` directives and multiple listening IP address-port combinations within one NGINX Plus `server` block. For more information on using `Host` and the `server_name` directive instead of IP addresses, see Server names at **nginx.org**.

Note: All IP addresses and names of objects (`upstream` blocks, virtual servers, pools, and so on) in this guide are examples only. Substitute the values from your BIG-IP LTM configuration.

Virtual Servers

As mentioned above, virtual servers are the primary listeners for both BIG-IP LTM and NGINX Plus, but the configuration syntax for defining them is quite different. Here, a virtual server at 192.168.10.10 listens on port 80 for HTTP traffic, and distributes incoming traffic between the two backend application servers listed in the `test_pool` upstream group.

BIG-IP LTM

```
# create pool test_pool members add { 10.10.10.10:80 10.10.10.20:80 }
# create virtual test_virtual { destination 192.168.10.10:80 pool
test_pool source-address-translation { type automap } ip-protocol
tcp profiles add { http } }
# save sys config
```

NGINX Plus

```
http {
    upstream test_pool {
        server 10.10.10.10:80;
        server 10.10.10.20:80;
    }

    server {
        listen 192.168.10.10:80;
        location / {
         proxy_pass http://test_pool;
        }
        #...
    }
}
```

Directive documentation: `listen`, `location`, `proxy_pass`, `server` (virtual), `server (upstream)`, `upstream`

SSL/TLS Offload (Termination and Proxy)

Handling SSL/TLS termination is a common use case for ADC load balancers. F5 BIG-IP LTM uses a proprietary SSL/TLS implementation. NGINX Plus relies on system libraries, so the version of OpenSSL is dictated by the OS. On BIG-IP LTM, a profile for each SSL/TLS key and certificate pair is attached to a virtual server (either as a client profile for encrypting traffic to and from the client, a server profile for encrypting backend traffic, or both). On NGINX Plus, the `ssl_certificate` and `ssl_certificate_key` directives are included in the server `block` for the virtual server.

There are two methods for handling SSL/TLS traffic on a load balancer instance, termination and proxying:

- With SSL/TLS termination, the load balancer and client communicate in an encrypted HTTPS session, in the same way a secure application like a banking website handles client encryption with SSL/TLS certificates. After decrypting the client message (effectively terminating the secure connection), the load balancer forwards the message to the upstream server over a cleartext (unencrypted) HTTP connection. In the other direction, the load balancer encrypts the server response before sending it to the client. SSL/TLS termination is a good option if the load balancer and upstream servers are on a secured network where there's no danger of outside agents intercepting and reading the cleartext backend traffic, and where upstream application performance is paramount.
- In the SSL/TLS proxy architecture, the load balancer still decrypts client-side traffic as it does in the termination model, but then it re-encrypts it before forwarding it to upstream servers. This is a good option where the server-side network is not secure or where the upstream servers can handle the computational workload required for SSL/TLS encryption and decryption.

BIG-IP LTM

- SSL/TLS Termination and Proxy: Creating SSL/TLS Virtual Server and Pool Members

```
# create pool ssl_test_pool members add { 10.10.10.10:443
10.10.10.20:443 }
# create virtual test_ssl_virtual { destination 192.168.10.10:443
pool ssl_test_pool source-address-translation { type automap }
ip-protocol tcp profiles add { http } }
# save /sys config
```

- SSL/TLS Termination: Creating a Client SSL/TLS Profile

```
# create profile client-ssl test_ssl_client_profile cert test.crt
key test.key
# modify virtual test_ssl_virtual profiles add { test_ssl_client_
profile }
# save /sys config
```

- SSL/TLS Proxy: Creating a Server SSL/TLS Profile

```
# create profile server-ssl test_ssl_server_profile cert test.crt
key test.key
# modify virtual test_ssl_virtual profiles add { test_ssl_server_
profile }
# save /sys config
```

NGINX Plus

- SSL/TLS Termination

```
upstream ssl_test_pool {
    server 10.10.10.10:443;
    server 10.10.10.20:443;
}

server {
    listen 192.168.10.10:443 ssl;
    ssl_certificate     /etc/nginx/ssl/test.crt;
    ssl_certificate_key /etc/nginx/ssl/test.key;

    location / {
    proxy_pass http://ssl_test_pool;
    }
}
```

- SSL/TLS Proxy

```
upstream ssl_test_pool {
    server 10.10.10.10:443;
}
server {
    listen 192.168.10.10:443 ssl;
    ssl_certificate     /etc/nginx/ssl/test.crt;
    ssl_certificate_key /etc/nginx/ssl/test.key;

    location / {
        proxy_pass https://ssl_test_pool;
        proxy_ssl_certificate /etc/nginx/ssl/client.pem;
        proxy_ssl_certificate_key /etc/nginx/ssl/client.key;
        proxy_ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
        proxy_ssl_ciphers HIGH:!aNULL:!MD5;
        proxy_ssl_trusted_certificate /etc/nginx/ssl/trusted_ca_
        cert.crt;
        proxy_ssl_verify on;
        proxy_ssl_verify_depth 2;
    }
}
```

Directive documentation: `listen`, `location`, `proxy_pass`, `proxy_ssl*`, `server (virtual)`, `server (upstream)`, `ssl_certificate and ssl_certificate_key`, `upstream`

Session Persistence

F5 BIG-IP LTM and NGINX Plus handle session persistence (also referred to as affinity) in a similar way and configure it at the same level: on the upstream server (BIG-IP LTM pool or NGINX Plus upstream block). Both support multiple forms of persistence. Session persistence is critical for applications that are not stateless and is helpful for continuous delivery use cases.

Cookie-Based Session Persistence

One method that is simple to configure and handles failover well for NGINX Plus, if compatible with the application, is sticky cookie. It works just like the cookie insert method in BIG-IP LTM: the load balancer creates a cookie that represents the server and the client then includes the cookie in each request, effectively offloading the session tracking from the load balancer itself.

- BIG-IP LTM: HTTP Cookie Persistence

```
# create persistence cookie test_bigip_cookie cookie-name
BIGIP_COOKIE_PERSIST expiration 1:0:0
# modify virtual test_virtual { persist replace-all-with {
test_bigip_cookie } }
# save /sys config
```

- BIG-IP LTM: HTTPS Cookie Persistence

```
# create persistence cookie test_bigip_cookie cookie-name
BIGIP_COOKIE_PERSIST expiration 1:0:0
# modify virtual test_ssl_virtual { persist replace-all-with {
test_bigip_cookie } }
# save /sys config
```

- NGINX Plus: HTTP Cookie Persistence

```
upstream test_pool {
    server 10.10.10.10:80;
    server 10.10.10.20:80;
    sticky cookie mysession expires=1h;
}
```

- NGINX Plus: HTTPS Cookie Persistence

```
upstream ssl_test_pool
    server 10.10.10.10:443
    server 10.10.10.20:443;
    sticky cookie mysession expires=1h;
}
```

Directive documentation: `server`, `sticky cookie`, `upstream`

Source IP Address-Based Session Persistence

Another form of session persistence is based on the source IP address recorded in the request packet (the IP address of the client making the request). For each request the load balancer calculates a hash on the IP address, and sends the request to the backend server that is associated with that hash. Because the has for a given IP address is always the same, all requests with the hash go to the same server. (For more details on the NGINX Plus implementation, see Choosing an NGINX Plus Load Balancing Technique on our blog).

- BIG-IP LTM

```
# modify virtual test_virtual { persist replace-all-with {source_
addr} }
# save /sys config
```

- NGINX Plus

```
upstream test_pool {
    ip_hash;
    server 10.10.10.10:80;
    server 10.10.10.20:80;
}
```

Directive documentation: ip_hash, server, upstream

Token-Based Session Persistence

Another method for session persistence takes advantage of a cookie or other token created within the session by the backend server, such as a **`jsessionid`**. To manage **`jsessionid`** creation and tracking, NGINX Plus creates a table in memory matching the cookie value with a specific backend server.

- BIG-IP LTM

  BIG-IP LTM does not natively support a learned (or universal) persistence profile without creating a more advanced iRule, which is out of scope for this document.

- NGINX Plus

```
upstream test_pool {
    server 10.10.10.10:80;
    server 10.10.10.20:80;
    sticky learn create=$upstream_cookie_jsessionid
                 lookup=$cookie_jsessionid
                 zone=client_sessions:1m;
}
```

Directive documentation: server, sticky learn, upstream

Keepalive Connections

Typically, a separate HTTP session is created and destroyed for every connection. This can be fine for short-lived connections, like requesting a small amount of content from a web server, but it can be highly inefficient for long-lived connections. Constantly creating and destroying sessions and connections can create high load for both the application server and client, slowing down page load and hurting the overall perception of the website or application's performance. HTTP keepalive connections, which instruct the load balancer to keep connections open for a session, are a necessary performance feature for web pages to load more quickly.

- BIG-IP LTM

```
# modify virtual test_virtual profiles add { oneconnect }
# modify virtual test_ssl_virtual profiles add { oneconnect }
# save /sys config
```

- NGINX Plus

```
upstream test_pool {
    server 10.10.10.10:80;
    server 10.10.10.20:80;
    keepalive 32;
}
```

Directive documentation: keepalive, server, upstream

Monitors (Health Checks)

F5 BIG-IP LTM uses the term *monitor* to refer to the process of verifying that a server is functioning correctly, while NGINX Plus uses *health check*. In an BIG-IP LTM configuration, the monitor is associated directly with a pool and applied to each node in the pool, whereas NGINX Plus places the health check in a **`location`** block.

The **`interval`** argument to the following BIG-IP LTM **`create`** command configures BIG-IP LTM to check the server every 5 seconds, which corresponds to the default frequency for NGINX Plus. NGINX Plus does not need the BIG-IP LTM **`timeout`** parameter as it implements the timeout function with the **`interval`** and **`fails`** parameters.

Note: This BIG-IP LTM configuration is for HTTP. For HTTPS, substitute **`test_ssl_monitor`** for **`test_monitor`** in both the **`create`** and **`modify`** commands. The same NGINX Plus configuration works for both HTTP and HTTPS.

- BIG-IP LTM

```
# create monitor http test_monitor defaults-from http send
"GET /index.html HTTP/1.0\r\n\r\n" interval 5 timeout 20
# modify pool test_pool monitor test_monitor
# save /sys config
```

- NGINX Plus

```
upstream test_pool {
    # ...
    zone test_pool_zone 64k;
}

server {
    # ...
    location / {
        proxy_pass http://test_pool;
        health_check interval=5 fails=2;
    }
}
```

Directive documentation: `health_check`, `location`, `proxy_pass`, `server`, `upstream`, `zone`

Summary of Converted Load Balancer Configuration

Here we put together the configuration entities, combining everything required to build a basic F5 BIG-IP LTM basic environment and detail how to migrate the same configuration to NGINX Plus.

BIG-IP LTM

```
# create pool test_pool members add { 10.10.10.10:80 10.10.10.20:80 }
# create virtual test_virtual { destination 192.168.10.10:80 pool
  test_pool source-address-translation { type automap } ip-protocol
  tcp profiles add { http } }
# create pool ssl_test_pool members add { 10.10.10.10:443
  10.10.10.20:443 }
# create virtual test_ssl_virtual { destination 192.168.10.10:443
  pool ssl_test_pool source-address-translation { type automap }
  ip-protocol tcp profiles add { http } }
# create profile client-ssl test_ssl_client_profile cert test.crt
  key test.key
# modify virtual test_ssl_virtual profiles add { test_ssl_client_
  profile }
# create profile server-ssl test_ssl_server_profile cert test.crt
  key test.key
# modify virtual test_ssl_virtual profiles add { test_ssl_server_
  profile }
# create persistence cookie test_bigip_cookie cookie-name BIGIP_
  COOKIE_PERSIST expiration 1:0:0
# modify virtual test_virtual { persist replace-all-with { test_
  bigip_cookie } }
# modify virtual test_ssl_virtual { persist replace-all-with
  { test_bigip_cookie } }
# modify virtual test_virtual profiles add { oneconnect }
# modify virtual test_ssl_virtual profiles add { oneconnect }
# create monitor http test_monitor defaults-from http send
  "GET /index.html HTTP/1.0\r\n\r\n" interval 5 timeout 20
# modify pool test_pool monitor test_monitor
# create monitor https test_ssl_monitor defaults-from https send
  "GET /index.html HTTP/1.0\r\n\r\n" interval 5 timeout 20
# modify pool ssl_test_pool monitor test_ssl_monitor
# save /sys config
```

NGINX Plus

The following configuration includes three additional directives which weren't discussed previously. Adding them is a best practice when proxying traffic:

- The `proxy_set_header Host $host` directive ensures the `Host` header received from the client is sent with the request to the backend server.
- The `proxy_http_version` directive sets the HTTP version to 1.1 for the connection to the backend server.
- The `proxy_set_header Connection ""` directive clears the `Connection` header so that NGINX Plus can keep maintain encrypted keepalive connections to the upstream servers.

We are also enabling live activity monitoring in the final `server` block. Live activity monitoring is implemented in the NGINX Plus API module and is exclusive to NGINX Plus. The wide range of statistics reported by the API is displayed on the built-in dashboard and can also be exported to any application performance management (APM) or monitoring tool that can consume JSON-formatted messages. For more detail on logging and monitoring see the NGINX Plus Admin Guide.

```
upstream test_pool {
    zone test_pool_zone 64k;
    server 10.10.10.10:80;
    server 10.10.10.20:80;
    sticky cookie mysession expires=1h;
    keepalive 32;
}

upstream ssl_test_pool {
    zone ssl_test_pool_zone 64k;
    server 10.10.10.10:443;
    server 10.10.10.20:443;
    sticky cookie mysession expires=1h;
    keepalive 32;
}
```

(continues)

```
server {
    listen 192.168.10.10:80 default_server;
    proxy_set_header Host $host;

    location / {
        proxy_pass http://test_pool;
        health_check;
        proxy_http_version 1.1;
    }

    location ~ /favicon.ico {
        root /usr/share/nginx/images;
    }
}

server {
    listen 192.168.10.10:443 ssl default_server;
    ssl_certificate     test.crt;
    ssl_certificate_key test.key;
    proxy_set_header    Host $host;

    location / {
        proxy_pass https://ssl_test_pool;
        proxy_http_version 1.1;
        proxy_set_header Connection “”;
        health_check;
    }

    location ~ /favicon.ico {
        root /usr/share/nginx/images;
    }
}
```

(continues)

```
server {
    listen 8080;
    status_zone status-page;
    root /usr/share/nginx/html;

    location /api {
        api write=on;
        # directives controlling access, such as ‘allow’ and ‘deny’
    }

    location = /dashboard.html {
        root /usr/share/nginx/html;
    }

    # Redirect requests made to the old (pre-R14) dashboard
    location = /status.html {
        return 301 /dashboard.html;
    }

    location ~ /favicon.ico {
        root /usr/share/nginx/images;
    }
}
```

4 Migrate F5 iRules to NGINX Plus

"Because we are dealing with a firehose of data from sources like Microsoft, Facebook, and Google, we needed to come up with a different solution in order to process that amount of traffic. Traditional hardware load balancers are expensive, and they lacked the flexibility and scale we needed."

–AJ Wilson, Vice President of Operations at IgnitionOne

Administrators of traditional hardware application delivery controllers (ADCs) are often faced with use cases that the rule sets provided by ADC vendors don't cover, and certainly not at the necessary fine-grained level of control over how requests and responses are processed. As a result, most organizations have written their own custom rules, and as they move to software-based load balancers, whether on premises or in the cloud, they need an easy way to migrate that ADC logic along with their applications. NGINX and NGINX Plus offer a powerful configuration language that covers many of these use cases.

This chapter explains how to convert several common types of policies or scripts from popular hardware ADCs to NGINX configuration blocks. We here at NGINX, Inc. have found that many times the apparent difficulty of converting policies is just due to differences in terminology or implementation and we'd like to bridge that gap.

Migrating Logic for Request Routing

Request routing, also known as content-based routing or content switching, is a way to host many different applications at the same fully qualified domain name (FQDN) but give the end user the impression of a single unified application. The ADC or NGINX Plus sends each request to the appropriate upstream application based on a header or the URI.

NGINX Plus implements request routing with the location directive, using either a URI prefix or regular expressions to match against requests. For more detail, see the NGINX Plus Admin Guide.

Example 1 – Request Routing Based on URI

In this example, request routing is based on the URI. Specifically, if the URI starts with **/music**, the request is routed to the **music_backend** upstream group. All other URIs are sent to the configured default upstream group.

F5 iRule

```
when HTTP_REQUEST {
    switch -glob [string tolower [HTTP::uri]] {
        "/music*"  {pool music_backend}
    default {pool default_backend}
    }
}
```

NGINX Plus Configuration

```
location /music {
    proxy_pass http://music_backend;
}
location / {
    proxy_pass http://default_backend;
}
```

Example 2 – Request Routing Based on the User-Agent Header

The following examples perform request routing based on the `User-Agent` header in the client request, directing requests from iOS and Android devices to separate backend application servers and traffic from all other devices to a configured application server. In the NGINX Plus `map` block, the `$http_user_agent` variable is compared to the specified regular expressions, and the `$upstream_choice` variable is set to the upstream group associated with the matching expression. Then the `$upstream_choice` variable determines which upstream group is chosen by the `proxy_pass` directive.

F5 iRule

```
when HTTP_REQUEST {
    switch -glob [HTTP::header User-Agent] {
        "*iPhone*"  -
        "*iPad*"  -
        "*iPod*"  {pool iosapp}
        "*Android*"  {pool androidapp}
        default {pool musicapp}
    }
}
```

NGINX Plus Configuration

```
map $http_user_agent $upstream_choice {
    ~(iPhone|iPad|iPod) ios_backend;
    ~Android android_backend;
    default default_backend;
}

server {
    listen 80;
    location / {
        proxy_pass http://$upstream_choice;
    }
}
```

Migrating Logic for Request Redirect

It is often necessary to redirect client requests, for example redirecting a client who sends a plain HTTP request to a connection secured with HTTPS. Other example use cases are shortened URLs or changes in the application URL structure.

To redirect requests with NGINX Plus, use the `return` directive. It takes two parameters: the response code (for example, `301` or `302`) and the redirect URL. For further discussion and more examples, see the NGINX Plus Admin Guide and Creating NGINX Rewrite Rules on the NGINX blog.

Example – Request Redirect to HTTPS

The following examples redirect the client from HTTP to HTTPS to ensure the session is encrypted.

F5 iRule

```
when HTTP_REQUEST {
    HTTP::redirect "https://[getfield [HTTP::host] ":" 1][HTTP::uri]"
}
```

NGINX Plus Configuration

```
location / {
    return 301 https://$host$request_uri;
}
```

Migrating Logic for Request Rewrite

Common reasons to rewrite requests are to provide additional information in headers or to change the URI, effectively hiding the directory structure of an application or creating URLs that are easier to read and remember.

To manipulate requests with NGINX Plus, use the `rewrite` directive. It takes two required parameters: a regular expression that matches the string to be rewritten and the replacement string. For further discussion and more examples, see the NGINX Plus Admin Guide and Creating NGINX Rewrite Rules on our blog.

Example – Request Rewrite to Change URI Structure

The following examples rewrite the URI structure of requests for **/music*/artist/song*** to **/mp3*/artist-song*.mp3**.

F5 iRule

```
when HTTP_REQUEST {
    if {[string tolower [HTTP::uri]] matches_regex {^/music/([a-z]+)/
    ([a-z]+)/?$} } {
        set myuri [string tolower [HTTP::uri]]
        HTTP::uri [regsub {^/music/([a-z]+)/([a-z]+)/?$} $myuri
        "/mp3/\1-\2.mp3"]
    }
}
```

NGINX Plus Configuration

```
location ~*^/music/[a-z]+/[a-z]+/?$ {
    rewrite ^/music/([a-z]+)/([a-z]+)/?$ /mp3/$1-$2.mp3 break;
    proxy_pass http://music_backend;
}
```

Migrating Logic for Response Rewrite

Rewriting the response body is often done along with request routing, to change links in the response body to reflect the new URI structure created by the request routing. It can also be used to make other changes to the response body before it's sent to the client.

To rewrite HTTP responses with NGINX Plus, use the `sub_filter` directive. It takes two parameters: the string for NGINX Plus to search for and replace, and the replacement string. For further discussion, see the NGINX Plus Admin Guide.

Example – Response Rewrite to Change Link Paths

The following examples search through the response body for links containing the **/mp3/** directory element and replace it with **/music/**.

F5 iRule

```
when HTTP_RESPONSE {
    if {[HTTP::header value Content-Type] contains "text"}{
        STREAM::expression {@/mp3/@/music/@}
        STREAM::enable
    }
}
```

NGINX Plus Configuration

```
location / {
    sub_filter '/mp3/' '/music/';
    proxy_pass http://default_backend;
}
```

Conclusion

As you can see, the NGINX and NGINX Plus configuration language covers many scripting use cases associated with ADCs. With this powerful toolset, migrating to software-based load balancing is simple and straightforward to accomplish.

Use cases that are more complicated, or require custom behavior can be accomplished with embedded scripting languages such as Lua, Perl, and JavaScript (with the NGINX JavaScript module in NGINX Plus R10 and later). For more information and use instructions, see Dynamic Modules with NGINX Plus.

5 Performance and Security Optimizations

"The amount of traffic that NGINX Plus can handle is unreal – even beyond our needs"

–James Ridle, IT Operations Manager at Montana Interactive

NGINX offers many performance and security optimizations, most of which are not available in Cisco ACE. In this chapter we show how configure some of the optimizations after you finish migrating from Cisco ACE.

Redirecting All Traffic to HTTPS

Google and other security-focused entities recommend using SSL for all Internet-facing site traffic. This prevents a man-in-the-middle from intercepting traffic and possibly monitoring users or swapping in its own content instead of your pages. Google puts weight behind its recommendation by rewarding sites that encrypt all traffic with higher search rankings. Beginning in July 2018, Google Chrome version 68 will mark all HTTP sites as "not secure".

The first `server` block listens on port 80 forces clients to use HTTPS, by returning status code `301 (Moved Permanently)` to redirect clients that have sent requests on port 80 to the same URL but with **https** as the scheme instead of **http**. The second virtual server listens on port 443 for the SSL-encrypted traffic:

```
server {
    listen 80 default_server;
    server_name www.example.com;
    return 301 https://$server_name$request_uri;
}

server {
    listen 443 ssl default_server;
    server_name www.example.com;
    ssl_certificate cert.crt;
    ssl_certificate_key cert.key;

    location / {
        proxy_set_header Host $host;
        proxy_pass http://SF_WEB;
    }
}
```

Enabling HTTP/2

HTTP/2 is the latest version of the HTTP protocol, standardized in 2015. Because it can pipeline requests and uses smaller HTTP headers, it performs better than HTTP/1.*x* in many situations. NGINX can act as an "HTTP/2 gateway" translating HTTP/2 back to HTTP/1.1 when communicating with real servers that do not support HTTP/2. HTTP/2 requires SSL.

To enable HTTP/2 support, add **`http2`** to the listen directive of an SSL-enabled virtual server.

```
server {
    listen 443 ssl http2 default_server;
    server_name www.example.com;
    ssl_certificate cert.crt;
    ssl_certificate_key cert.key;
}
```

Enabling Content Caching

Some of the world's largest CDNs, including MaxCDN, Fastly, and Instart Logic, take advantage of NGINX content caching. When Netflix decided to build its own CDN, it used NGINX open source at its core. In fact, it's estimated that nearly a third of North America's internet traffic flows through NGINX caches due to Netflix's popular streaming service. You can use NGINX caching to improve performance for both dynamic and static content.

```
proxy_cache_path /tmp/cache levels=1:2
        keys_zone=my_cache:10m max_size=10g
        inactive=60m use_temp_path=off;
server {
    location / {
        proxy_cache my_cache;
        proxy_set_header Host $host;
    proxy_pass http://SF_WEB;
    }
}
```

The **proxy_cache_path** directive sets the path and configuration of the cache, and the **proxy_cache** directive activates it:

- **proxy_cache_path** – Defines a 10 GB cache stored in **/tmp/cache**. Inactive elements in the cache are deleted after 60 minutes.
- **proxy_cache** – Enables caching for the location block it is in. In the example, it applies to all content on the site.

For more details on NGINX content caching, see A Guide to Caching with NGINX and NGINX Plus on our blog.

Enabling Keepalive Connections to the Server Farm

For maximum compatibility with applications, by default NGINX makes HTTP/1.0 connections to the real servers in the server farm. If the real servers support HTTP/1.1 (most do), we recommend instructing NGINX to use it. HTTP/1.1 supports keepalive TCP connections, and reusing connections for multiple requests improves performance by reducing the overhead of creating new TCP connections.

```
upstream SF_WEB {
    server 10.10.50.10:80; # content server web-one
    server 10.10.50.11:80; # content server web-two
    keepalive 32;
}
server {
    location / {
        proxy_http_version 1.1;
        proxy_set_header Connection "";
        proxy_set_header Host $host;
        proxy_pass http://my_upstream;
    }
}
```

Explanation of the key directives:

- **`keepalive`** – Enables the TCP connection cache and sets the maximum number of TCP connections to keep open (here, 32).
- **`proxy_http_version`** – Upgrades connections to the server farm to HTTP/1.1.
- **`proxy_set_header`** – Enables keepalive TCP connections by clearing the Connection: Close HTTP header. By default NGINX uses HTTP/1.0 with `Connection: Close`.

6 AppNexus Case Study

"Moving to the next generation of F5 hardware was going to cost more than $1M per data center. NGINX Plus gave us 50% more transactions per server, for one-sixth the price. We're now 100% hardware free."

–Senior Networking Leader, AppNexus

Overview

AppNexus' mission is to create a better internet. They harness data and machine learning to power the world's digital audience platforms. Their real-time decisioning platform supports core products that enable their customers to acquire, engage, and monetize their audience.

Challenge

Internet advertising helps companies to reach their target audiences, creators to get the most from their content, and consumers to get access to online experiences. Placing and presenting the right ads to the right viewer at the right time is incredibly complex. Many ads are dynamically selected for individual users through a real-time bidding process that occurs within milliseconds behind the scenes. And that bidding runs on AppNexus infrastructure. AppNexus is the world's leading independent ad tech company. It provides an open, powerful programmatic platform that empowers its customers – including some of the world's most influential advertising and media companies – to more effectively buy and sell ads. Today, AppNexus relies on NGINX Plus to handle some of its most challenging customer use cases.

AppNexus customers buy and sell ad impressions through a real-time bidding (RTB) application. As a web page loads, each available impression on the page triggers an HTTP retrieval to an AppNexus server, which hosts the ads. Advertisers bid to have their ad served to the user, and the winner's ad is delivered to the user's page. The entire process takes just a few hundred milliseconds, but it occurs billions of times each day.

AppNexus customers also include search engine companies and specialists who aggregate web page impressions from mobile devices. These supply-side partners (SSPs) also use the AppNexus RTB capabilities, but deliver ads from their own servers instead of having AppNexus host the ad content. The extra step of fetching an ad from the SSP adds time to the process. Although there are far fewer SSPs bidding for ad slots than individual advertisers, which means there are fewer connections, their HTTP sessions stay open much longer. That makes load balancing and capacity scaling of SSP requests much more challenging.

Founded
2007

Headquarters
New York, New York

Overview
AppNexus is a technology company that provides trading solutions and powers market-places for Internet advertising.

Challenge
Capacity scaling of supply-side partner (SSP) requests, manage real-time bidding (RTB) app traffic, and manage availability of indivi-dual software load balancers.

Results
- **Cost savings** – NGINX Plus is ⅙ of the price of previous F5 BIG-IP
- **Better performance** – 50% more transactions per server compared to F5 BIG-IP
- **Operational insight** – Deep NGINX Plus statistics accelerate troubleshooting and perfor-mance tuning

AppNexus data centers are located around the world, and each data center houses clusters of RTB application servers. Incoming traffic is load balanced first across the data centers, based on network proximity to the client, and then load balanced across the servers in the cluster.

"To improve load balancing for our SSP requests, we first tried a traditional hardware appliance," says Louis Mamakos, Senior Network Architect for AppNexus. "It improved load balancing, but the hardware costs negatively affected our cost of goods sold. So we started looking for software solutions."

AppNexus considered HAProxy and Apache HTTP Server for reverse proxying but had concerns about their multithreading scaling abilities. Auctions typically run on 12-core Intel Xeon CPUs. Because AppNexus also offers cloud services hosted on these same systems, taking advantage of their multi-threading capability makes most efficient use of the available computing power. AppNexus already used open source NGINX for some content delivery.

After the company discovered NGINX Plus load balancing SSP requests solutions, they were sold.

Solutions

NGINX is the world's most popular open source web server for high-traffic websites, and NGINX Plus adds advanced load balancing, session persistence, live activity monitoring, and other features to provide reliability, control, and consistent performance for HTTP applications. AppNexus initially deployed three instances of NGINX Plus to manage RTB application traffic, combined with a DNS round-robin load balancer to manage availability of individual software load balancers. NGINX Plus runs on Dell PowerEdge C6220 systems with 12-core Intel Xeon E5-2630L CPUs running at 2.0GHz. These systems are configured with dual, bonded 10-GbE interfaces.

AppNexus automates configuration of its NGINX Plus solution using a configuration database and Puppet. Automation is critical to the company's success, enabling it to efficiently scale out while keeping costs under control.

> *"NGINX Plus does exactly what we need it to do. We can terminate each HTTP connection and process each request individually to apply the best load-balancing policy. The software also can automatically optimize HTTP transactions to keep them alive, improving performance. We're sustaining approximately 75,000 transactions per second through a single instance of NGINX Plus."*
>
> – Louis Mamakos, Senior Network Architect at AppNexus

Results

Health Checks Maintain High Availability

One of the main reasons that AppNexus chose NGINX Plus was for its active application health checks. AppNexus performs rolling upgrades on its servers, meaning that at any given time, a group of servers might be out of service for maintenance. In the past, requests directed to unavailable servers had to be retried, which increased the risk of missing the ad placement window. NGINX Plus continually checks the health of upstream servers and does not send requests to servers that are offline. When maintenance is complete and the team places the server back in service, the health check succeeds, and NGINX Plus gradually reintroduces the server into the load-balanced cluster using a slow-start feature. NGINX Plus health checks help ensure that customers don't miss ad impression opportunities when servers are taken down for maintenance.

Detailed Statistics Simplify Operational Insight

With thousands of servers generating more than one million data points per minute, ad hoc reporting is impossible. Mamakos says request logging is impractical because within minutes he's drowning in log files. Instead, the team relies on automated statistics from NGINX Plus, which are fed into AppNexus' Graphite real-time graphing system. From there, other solutions extract and analyze data.

"The detailed statistics that NGINX Plus delivers help us monitor the software and characterize overall performance," says Mamakos. "We gain visibility into how NGINX Plus sees requests and activity. By knowing how NGINX Plus sees things, we can accelerate troubleshooting, continuously tune performance, and better align our systems to handle whatever load-balancing needs come along."

Cost Effectiveness Improves the Bottom Line

Even with ultra-high transaction volumes, acceptable margins demand cost-effective solutions. With NGINX Plus, AppNexus gained a highly cost-effective way to improve SSP request performance and maintain a low cost of goods sold. Automation also saves time and frees the AppNexus team to focus on innovation. As time goes on, AppNexus can scale its capabilities while maximizing its staff's time.

Opportunities Abound

Since initially deploying NGINX Plus in 2014 for one SSP partner, AppNexus has expanded its use to additional partners. In addition, Lua programming features will allow AppNexus to gain even deeper visibility into statistics and to customize functions, such as packet inspection, without having to build special tools.

"As our company and this market mature, we're increasingly focusing on optimizing and driving efficiency," says Mamakos. "NGINX Plus offers us numerous ways to uncover opportunities and monetize efficiency. It's really an engine for addressing future customer services and growing our business."

About AppNexus

AppNexus is a technology company that provides trading solutions and powers marketplaces for Internet advertising. Its open, unified, and powerful programmatic platform empowers customers to more effectively buy and sell media, allowing them to innovate, differentiate, and transform their businesses. As the world's leading independent ad tech company, AppNexus is led by the pioneers of the web's original ad exchanges. Headquartered in New York City with 23 global offices, AppNexus employs more than 1000 of the brightest minds in advertising and technology who believe that advertising powers the Internet. For more information, visit www.AppNexus.com.

Appendix A:

Document Revision History

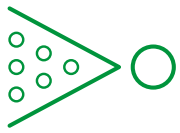
Version	Date	Description
1.0	2018-11-29	Initial release

Try NGINX Plus and NGINX WAF free for 30 days

Replace F5 BIG-IP with NGINX Plus and gain the freedom to innovate without being constrained by hardware.

Cost Savings

Over 80% cost savings compared to F5 BIG-IP, with all the performance and features you expect.



Reduced Complexity

The only all-in-one load balancer, content cache, web server, and web application firewall helps reduce infrastructure sprawl.

Exclusive Features

JWT authentication, high availability, the NGINX Plus API, and other advanced functionality are only available in NGINX Plus.

NGINX WAF

A trial of the NGINX WAF, based on ModSecurity, is included when you download a trial of NGINX Plus.

Download at **nginx.com/freetrial**

NGINX